Санкт-Петербургский государственный университет

Прикладная математика и информатика

Исследование операций и принятие решений в задачах оптимизации, управления и экономики

Агеева Надежда Михайловна

Поиск цепных кодов наибольшей длины

Выпускная квалификационная работа

Научный руководитель:

к. ф.-м. н., доцент Агафонова И. В.

Рецензент:

к. ф.-м. н., ст. преп. Соловьева Н. А.

Санкт-Петербург

2017

SAINT-PETERSBURG STATE UNIVERSITY

Applied Mathematics and Computer Science

Operation Research and Decision Making in Optimisation,

Control and Economics Problems

Ageeva Nadezhda

Search of maximum length circuit code

Graduation Project

Scientific supervisor:

Associate professor Agafonova I. V.

Reviewer:

Senior Lecturer Soloveva N. A.

Saint-Petersburg

2017

Содержание

1 Введение

Задача, известная как Snake-in-the-Box problem («змея в ящике») заключается в поиске самого длинного незамкнутого (цепь) или замкнутого (цикл) пути без самопересечений по вершинам гиперкуба Q_n. В настоящей работе рассматривается обобщение этой задачи, в котором ищутся цепи и циклы, где каждая пара вершин, находящихся на расстоянии не менее k в пути или цикле, также находится на расстоянии не менее k в Q_n.

Впервые данная задача была описана в связи с теорией кодов, исправляющих ошибки. Чем длинней цепь, тем больший алфавит можно закодировать, и чем больше k, тем выше вероятность обнаружения ошибок.

Для размерностей от $n = 1$ до $n = 7$ змеи максимальной длины были найдены с помощью полного перебора, но при $n > 7$ пространство решений становится слишком велико. В случае $n = 7$ цикл максимальной длины был найден K. Kochut в 2009 г. экономным полным перебором (раздел 5).

В связи с этим в работе ставятся и решаются следующие задачи:

- Дать обзор точных и приближенных методов построения и поиска цепей и циклов наибольшей длины в n-мерном гиперкубе.
- Представить современные оценки максимальных длин и попытаться получить цепи с длинами, максимально близкими к этим оценкам.
- Реализовать алгоритм исчерпывающего поиска канонических цепей, предложенный K. Kochut для цепей размаха $k = 2$. Изучить его расширения для $k > 2$.
- Исследовать возможности модификаций алгоритмов приближенного поиска цепей при больших n. Реализовать один из вариантов.
- Провести численные расчеты для $n = 5, 6, ..., 15$ и $k = 2, 3, ..., 9$.

- Изучить применение к данной задаче нескольких эвристических алгоритмов.
- Рассмотреть связь длинных цепей с кодами, исправляющими ошибки.

Все перечисленные цели были реализованы.

2 Основные определения

Введем определение цепных кодов.

Пусть I – множество из двух элементов, $I = \{0, 1\}$. $I^n = I \times ... \times I$. При этом будем называть элементы множества I *словами длины* n и обозначать их $x_i = (x_{i_1}, x_{i_2}, ..., x_{i_n})$, где $x_{i_j} \in I$.

Расстояние Хемминга в I^n вычисляется по формуле

$$\rho(x, y) = \sum_{i=1}^{n} |x_i - y_i|.$$

Конечная последовательность $x_0, x_1, ..., x_l$ двоичных слов длины n *образует цепной код с размахом* k, если выполняется два следующих условия:

1. $\rho(x_i, x_{i+1}) = 1$ при $i = 0, 1, ..., l - 1$;
2. для всех $0 \leq i, j \leq l$ из неравенства $|i-j| \geq k$ следует, что $\rho(x_i, x_j) \geq k$, где $\rho(x, y)$ – расстояние Хемминга между словами $x = (x_1, x_2, ..., x_n)$ и $y = (y_1, y_2, ..., y_n)$. При этом мы принимаем $l \geq k$, чтобы не брать во внимание вырожденные случаи.

Такой цепной код будем называть (n, k)-цепью длины l. Наибольшую длину такого кода будем обозначать $L(n, k)$.

При $d = 2$ цепь $(n, 2)$ называют «Snake-in-the-Box» или «змеей в ящике».

По определению цепному (n, k)-коду в графе единичного n-мерного

гиперкуба соответствует путь по ребрам этого гиперкуба, который ближе, чем на расстояние d, не подходит сам к себе.

Циклически замкнутый путь называется (n, k)-циклом (или «coil»).

Работа посвящена в первую очередь построению цепей.

3 Способы задания цепей и циклов

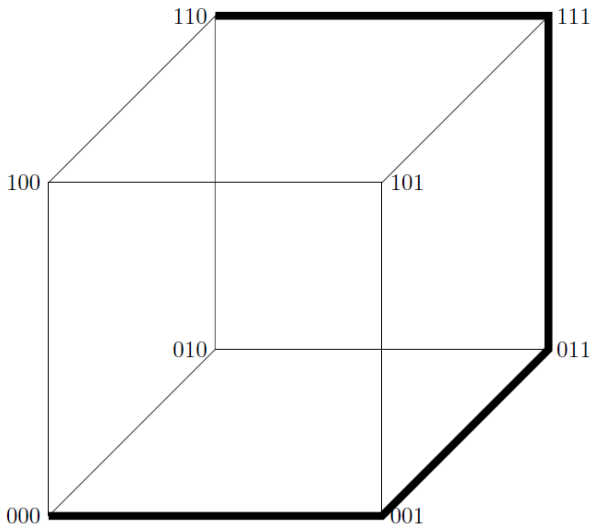

Рис. 1: Максимальная $(3, 2)$-цепь

Имеются два основных способа задания:

1. Цепь и цикл можно задавать в виде последовательности узлов. Узлы пронумерованы в порядке, соответствующем двоичным векторам.
2. Другой способ задания – переходная последовательность. Она представляет собой вектор чисел в диапазоне $0..n-1$, соответствующих номерам изменяющихся двоичных разрядов.

Например, как показано на Рис. 1, $(3, 2)$-цепь наибольшей длины 4 образует последовательность: (000), (001), (011), (111), (110).

Её можно задать таким образом: $\{0, 1, 3, 7, 6\}$. И иначе: $\{0, 1, 2, 0\}$.

В дальнейшем будем отождествлять вектор с числом, например $011 = 3$.

4 Постановка задачи

Требуется построить цепной код в соответствии с определением раздела 2, максимально возможной длины при заданных $n \geq 2$ и $k \geq 2$.

При малых n и k задача решается полным перебором всех возможных путей, но при их увеличении вычислительная сложность резко возрастает. Например, до публикации K. Kochut (2009 г.) для $n = 7$ и $k = 2$ применялись только приближенные методы.

5 Базовый алгоритм K. Kochut для k=2

Напомним, целью является максимизация длины последовательности, задающей цепной код.

Два узла являются *смежными*, если они различаются строго в одной координате.

Цепи *эквивалентны*, если они отличаются перестановкой координат, таким образом каждая цепь относится к классу из $n!$ цепей, которые в частности, все имеют одинаковую длину. То есть достаточно рассмотреть лишь одного представителя из каждого класса, тем самым пространство перебора уменьшается.

Для того, чтобы исследовать ровно одну цепь в каждом классе эквивалентности, будем рассматривать только цепи, которые являются *«каноническими»* в том смысле, что первое вхождение 1 в каждой позиции следует линейному порядку от наименее значимого до наиболее значимого. Следуя

этому правилу, для примера для $n = 7$, следующий узел после начала координат 0000000 должен быть 0000001, а не 1000000 или любой из других пяти узлов, расположенных рядом с началом координат.

Имеющаяся цепь продлевается за счет перемещения в соседний узел, который удовлетворяет второму условию из определения цепного кода. Для $k = 2$ его можно переформулировать так: добавляемый узел не должен являться смежным с любыми другими узлами в имеющейся цепи.

В случае поиска циклов K. Kochut предлагает *дополнительную оптимизацию* для усечения дерева поиска: исключаются цепи, которые никогда не могут быть замкнуты в циклы (путем возвращения к началу координат). Узлы, смежные с 0000000, будем называть *узлами возврата*.

Поскольку алгоритм определения цикла начинает нумерацию канонических цепей в начале координат, есть $n - 1$ узлов, которые в конечном итоге могут привести обратно к началу координат (один из n соседних узлов, например, 0000001, используется для первоначального удлинения цепи от начала координат и, следовательно, исключается в качестве потенциального узла возврата). Каждый раз, когда цепь удлиняется на один узел, все его соседи отмечаются, и если какие-либо из них также являются соседними с началом координат, они вычёркиваются из списка потенциальных узлов возврата. Очевидно, что, если число потенциальных узлов возврата падает до нуля, имеющаяся цепь может быть исключена, даже несмотря на то, что возможно ее дальнейшее удлинение.

6 Примеры, демонстрирующие работу алгоритма

Пример 1:

Построим цикл наибольшей длины для $n = 5$ и $k = 2$. Очередной узел, добавляемый в путь, будем выделять жирным шрифтом.

1. Начинаем всегда с начала координат: **00000** $= 0$.
2. Первое вхождение 1 в каждой позиции следует линейному порядку от наименее значимого до наиболее значимого, значит добавляем узел **00001** $= 1$. Возможными узлами возврата являются: 00010, 00100, 01000, 10000.
3. - Пойти в 00000 не можем, так как такой узел уже включён в цепь.
   - Поэтому идем в **00011** $= 3$. Этот узел является смежным с 00010, поэтому теперь список возможных узлов возврата теперь выглядит так: 00100, 01000, 10000.
4. - Пойти в 00010 не можем, так как это узел возврата (цепь замкнется, имея при этом не максимальную длину).
   - Поэтому идем в **00111** $= 7$.
5. Идем в **00110** $= 6$. Этот узел является смежным с 00100, поэтому теперь список возможных узлов возврата теперь выглядит так: 01000, 10000.
6. - Пойти в 00100 не можем, так как $\rho(00100, 00000) < 2$.
   - В 00010 аналогично: $\rho(00010, 00011) < 2$.
   - Идем в **01110** $= 14$.
7. - Пойти в 01111 не можем, так как $\rho(01111, 00111) < 2$.
   - Идем в **01100** $= 12$. Этот узел является смежным с 01000, поэтому

теперь список возможных узлов возврата теперь выглядит так: 10000. Он пока что не пуст, поэтому продолжаем.

8. Идем в **01101** $= 13$.
9. - Пойти в 01100 не можем, так как такой узел уже есть в цепи.
   - Пойти в 01111 не можем, так как $\rho(01111, 00111) < 2$.
   - Пойти в 01001 не можем, так как $\rho(01001, 00001) < 2$.
   - Пойти в 00101 не можем, так как $\rho(00101, 00001) < 2$.
   - Идем в **11101** $= 29$.
10. - Пойти в 11100 не можем, так как $\rho(11100, 01100) < 2$.
    - Идем в **11111** $= 31$.
11. - Пойти в 11110 не можем, так как $\rho(11110, 01110) < 2$.
    - Пойти в 11101 не можем, так как $\rho(11101, 01101) < 2$.
    - Идем в **11011** $= 27$.
12. Идем в **11010** $= 26$.
13. - Пойти в 11011 не можем, так как такой узел уже есть в цепи.
    - Идем в **11000** $= 24$.
14. - Пойти в 11001 не можем, так как $\rho(11001, 11011) < 2$.
    - Пойти в 11010 не можем, так как такой узел уже есть в цепи.
    - Пойти в 11100 не можем, так как $\rho(11100, 11101) < 2$.
    - Идем в **10000** $= 16$ – последний имеющийся узел возврата.
15. Возвращаемся в начало координат **00000** $= 0$.

Таким образом, получили цикл 0, 1, 3, 7, 6, 14, 12, 13, 29, 31, 27, 26, 24, 16, 0. Его длина – 14, она является максимальной. Нетрудно заметить, что если исключить 2 последние добавленные вершины, получится $(5, 2)$-цепь максимальной длины 13.

Как теперь получить один из $n!$ эквивалентных циклов? Пусть полученный в результате работы алгоритма цикл соответствует тождественной перестановке 12345. Запишем цикл, соответствующий перестановке 14325. Для этого поменяем 2-ю и 4-ю координаты узлов местами:

$00000 \to 00000 = 0$, $00001 \to 00001 = 1$, $00011 \to 01001 = 9$,

$00111 \to 01101 = 13$, $00110 \to 01100 = 12$, $01110 \to 01110 = 14$,

$01100 \to 00110 = 6$, $01101 \to 00111 = 7$, $11101 \to 10111 = 23$,

$11111 \to 11111 = 31$, $11011 \to 11011 = 27$, $11010 \to 11010 = 26$,

$11000 \to 10010 = 18$, $10000 \to 10000 = 16$, $00000 \to 00000 = 0$.

Как видно, этот цикл проходит через некоторые другие вершины 5-мерного гиперкуба, тем не менее, является эквивалентным. Как известно, перестановок длины n существует $n!$, поэтому число циклов в каждом классе эквивалентности в точности $n!$.

Пример 2:

Этот пример описан K.J. Kochut в работе [6]. Последовательность 0 1 3 7 15 13 12 28 30 26 27 25 57 56 40 104 72 73 75 107 111 110 46 38 36 52 116 124 125 93 95 87 119 55 51 50 114 98 66 70 68 69 101 97 113 81 80 16 0 задаёт цикл максимальной длины 48 в 7-мерном гиперкубе. Всего автором было найдено 12 максимальных неэквивалентных $(7, 2)$-циклов.

7 Модификация алгоритма K. Kochut для $k>2$

В настоящей работе представлен экономный алгоритм, осуществляющий поиск в глубину, который в n-мерном единичном гиперкубе Q_n исчерпывающе ищет длинные канонические k-цепи (согласно данному ниже определению). Это адаптация алгоритма K. Kochut для $k = 2$, которая требует нескольких нетривиальных изменений для общих k.

Цепь находится в *канонической форме*, если она начинается в $x_0 = 0$ и каждый старший двоичный разряд d может меняться лишь после того, как хотя бы единожды менялся каждый из младших разрядов.

Пример канонической $(5, 2)$-цепи, построенной ранее в примере раздела 6: {00000, 00001, 00011, 00111, 00110, 01110, 01100, 01101, 11101, 11111, 11011, ...}

Мы будем говорить, что вершина y *конфликтует с вершиной* x, если две вершины отличаются менее чем в k битовых позициях.

Основная идея алгоритма состоит в том, чтобы обновить список конфликтов с неиспользуемыми вершинами при расширении пути поиска. Это позволяет определить, может ли путь поиска быть расширен через заданную вершину x.

Структура данных, необходимая для поддержания конфликтов, – это массив $numConflicts[\,]$, где каждая ячейка $numConflicts[x]$ хранит количество вершин в текущем пути поиска α, которые конфликтуют с x. По определению каждая вершина конфликтует с $C_n^1 + ... + C_n^{k-1}$ другими вершинами в Q_n.

k-змея, которая ищется в каждом рекурсивном вызове метода $Search(d)$, хранится в $\alpha = \alpha_0, ..., \alpha_{t-1}$, где каждое α_i - целочисленное представление вершины гиперкуба Q_n. Параметр d указывает наибольший уже посещенный двоичный разряд в настоящее время.

На Рис. 2 приведен пвсевдокод алгоритма, использующий согласно предложениям S. Hood следующие обозначения:

- $numConflicts[x]$: количество вершин в α, конфликтующих с x
- $adj[x][d]$: сосед x, полученный изменением d-го разряда
- $conflict[x]$: полный набор вершин Q_n, конфликтующих с x

```
void Search (int d) {
int x; int y; int p ={последний элемент текущего пути}
    for (int i = 0; i ≤ min(d + 1, n − 1); i + +) {
        {x получен изменением i-го бита элемента p}
        if (numConflicts[x] < k) {
            α_t = x;
            for each y ∈ conflict[x] do numConflicts[y] + +;
            Search(max(i, d));
        }
    }
    for each y ∈ conflict[x] do numConflicts[y] − −;
    {удаление из пути последнего элемента}
}
```

Рис. 2: Псевдокод алгоритма раздела 7

Массивы adj и $conflict$ вычисляются заранее.

Для запуска алгоритма мы инициализируем $numConflicts[x] = 0$ для каждой вершины x. Затем строим начальный путь из первых $k + 1$ вершин: 0, $2^{1-1}, ..., 2^k - 1$, добавляя по одной в путь, одновременно увеличивая $numConflicts[i]$ для каждой вершины гиперкуба.

Исходным вызовом является $Search(k)$. Во время рекурсивного вызова для каждого соседа x последней вершины a_{t-1} проверяется, может ли он быть включен в путь без нарушения ограничений на k или d. Для того, чтобы соблюдалось ограничение на k, может быть не более $k - 1$ вершин (последние $k - 1$ вершин), которые конфликтуют с x. Если x не нарушает ограничений, мы увеличиваем $numConflicts[y]$ для каждой вершины y, которая конфликтует с x, а затем рекурсивно ищем более длинные k-змеи.

8 История и состояние вопроса в настоящее время

Впервые задача поиска змей в ящике была поставлена W. H. Kautz в работе «Unit-distance error-checking codes» в 1958 году.

Традиционно, математические подходы к $(n, 2)$-цепям (змеям в ящике) привлекли две основные стратегии. Первый способ – с использованием инструментов логики, дискретной математики и теории графов, в то время как второй анализирует структуру пространства решений, чтобы его уменьшить, а далее уже с помощью исчерпывающего алгоритма перебираются элементы оставшегося усеченного пространства. Эти методы были успешны при определении максимально длинных цепей и циклов в гиперкубах размерности до 7 включительно.

Для больших размерностей пространство поиска труднообрабатываемо, хотя математические подходы и были усилены за счет использования вычислительной техники. Для размерностей 8 и выше, даже с имеющимся в настоящее время оборудованием, исчерпывающий поиск остается непрактичным. Это открыло двери для менее традиционных эвристических методов вычислительного поиска. В их числе можно упомянуть генетический алгоритм (Walter D. Potter, Robert W. Robinson и другие в работе «Using the genetic algorithm to find snake-in-the-box codes» 1994), муравьиный алгоритм (Shilpa P. Hardas в работе «An ant colony approach to the snake-in-the-box problem» 2005), PBSHC (популяция, основанная на стохастическом поиске восхождением к вершине) (Daniel R. Tuohy, Darren A. Casella and Walter D. Potter в работе «Searching for snake-in-the-box codes with evolved pruning models»), алгоритм с дополнительной оптимизацией для ускоренного перебора (Kochut J. Krys в работе «Snake-in-the-box codes for dimension 7»).

9 Современные верхние и нижние оценки длин цепей и циклов

Оценки, приведенные в этом разделе, учитываются при построении цепей и циклов приближенными методами. Они полезны для определения, насколько найденная длина приближена к максимальной.

Известно, что $L(n,1) = 2^n - 1$. Чтобы в этом убедиться, достаточно построить в гиперкубе гамильтонову цепь. $(n,1)$-цепи называются ещё кодами Грея, они широко применяются.

Точные значения для $L(n,2)$ известны при $n < 9$. Для любого $n \geq 8$ самая современная на данный момент нижняя оценка: $L(n,2) \geq const \cdot 2^n$, где $const > 0.3105$.

Приведем известные на сегодняшний день результаты, опубликованные в работе [2] авторов David Allison and Daniel Paulusma:

- Для $n \leq 5$: W.H. Kautz определил самый длинный цикл («Unit-distance error-checking codes», 1958).
- Для $n \leq 6$:, используя исчерпывающий алгоритм поиска, D.W. Davies нашел самую длинную цепь, а также цикл для $n = 6$ («Longest «separated» paths and loops in an n cube», 1965).
- Для $n = 7$: W. Potter, J. Robinson, J. Miller and K.J. Kochut использовали генетический алгоритм для нахождения максимальной длины змеи («Using the genetic algorithm to find Snake- In-The-Box codes», 1994) и K.J. Kochut, как уже говорилось, использовал исчерпывающий поиск с дополнительной оптимизацией для цикла («Snake-In-The-Box codes for dimension 7», 1996).
- Для $n = 8$: P.R.J. Ostergard и V.H. Pettersson использовали канони-

ческое приращение в приложении и к цепям, и к циклам («Exhaustive search for snake-in-the-box codes», 2015 и «On the maximum length of coil-in-the-box codes in dimension 8», 2014).

Для $n \geq 9$: на данный момент известны лишь нижние границы. Наилучшие из них, имеющиеся сейчас:

- Для $n = 9$: E. Wynn доказал, что наибольшие длины цепи и цикла по крайней мере 190 и 188 соответственно («Constructing circuit codes by permuting initial sequences»).
- Для $n = 10$: D. Kinny показал, что змея имеет длину по крайней мере 370 («A new approach to the Snake-In-The-Box-problem», 2012), а S.J. Meyerson, T.E. Drapela, W.E. Whiteside and W.D. Potter показали, что для цикла эта цифра составляет 362 («Finding longest paths in hypercubes: 11 new lower bounds for snakes, coils, and symmetrical coils», 2015).
- Для $11 \leq n \leq 12$: результаты S.J. Meyerson и других соавторов с 2015 года, как известно, дают лучшие нижние оценки.
- Для $n = 13$: S.J. Meyerson и другие показали, что нижняя граница для змеи составляет по крайней мере 2520, а H. Abbott and M. Katchalski, что наиболее длинный цикл имеет длину 2468 («On the construction of Snake In The Box codes», 1991).
- Для $14 \leq n \leq 20$: результаты H. Abbott and M. Katchalski с 1991 года всё ещё не побиты.

Что касается верхних оценок, наилучшую из них на сегодняшний день получил G. Zemor:

$$L(n, 2) \leq 2^{n-1} \left(1 - \frac{1}{89n^{1/2}} + O\left(\frac{1}{n}\right)\right).$$

В заключение приведем лучшие известные оценки $L(n, k)$ при $k = 2t + 1$,

$t \geq 1$. Верхняя получена R.J. Douglas:

$$L(n, 2t+1) \leq \frac{2^n}{C_n^t - 2C_{n-1}^{t-1} + P(n)},$$

где $P(n)$ – некоторый полином от n степени t с коэффициентом при старшем члене $1/t!$. Нижняя получена А.А. Евдокимовым [1]:

$$L(n,k) \geq \frac{2^n}{n^{t(1+\varepsilon_n)}}, \text{ где } \varepsilon_n = \frac{2\log_2\log_2 n}{\log_2 n}.$$

10 Связь (n,k)-кодов с кодами, исправляющими ошибки

Из теории блочных помехоустойчивых кодов для нечетного $k = 2r+1$, имея (n,k)-цепь длины l, легко построить помехоустойчивый (n,k)-код мощности $\lfloor l/k \rfloor + 1$, исправляющий r ошибок. Для этого, двигаясь вдоль по (n,k)-цепи, будем включать в код каждую k-ю по порядку вершину, что и обеспечит минимальное кодовое расстояние $2r+1$.

Например, для $(9,3)$-цепи длины 57 построим $(9,3)$-код мощности $\lfloor 57/3 \rfloor + 1 = 20$, исправляющий 1 ошибку.

Сама цепь: 0 1 3 7 15 14 30 28 60 61 57 59 123 122 250 234 202 203 201 217 209 208 212 196 228 164 166 182 183 247 503 499 483 481 353 361 360 364 366 358 326 322 338 274 306 304 432 440 408 392 396 397 389 405 277 341 349 351.

Будем зачислять в код каждую 3-ю по порядку вершину цепи: 0 7 30 61 123 234 201 208 228 182 503 481 360 358 338 304 408 397 277 351.

Таким образом, список кодовых слов: 000000000, 000000111, 000011110, 000111101, 001111011 и так далее, всего 20 слов. Минимальное кодовое расстояние здесь составляет 3.

11 Обзор приближенных алгоритмов

В этой работе мы рассмотрим некоторые приближенные подходы к «охоте на змей». Один из них, п. 11.1, будет запрограммирован.

11.1 Обрезка ветвей

Обрезка ветвей – отказ от полного перебора путем исключения вершин, которые кажутся «неперспективными» по какому-либо критерию. Из критериев, предложенных D. Tuohy в [8], можно выделить следующие:

- *Плотность*: для каждого кандидата на добавление в цепь известно число вершин в пути, которые находятся от него на расстоянии k, эту величину назовем плотностью. Чем оно больше, тем плотней цепь намотана и тем больше узлов доступны на каждом шаге.

  В программе вычисляется минимальная и максимальная плотность кандидатов. Чем ближе плотность вершины к максимальной, тем с большей вероятностью она будет включена в цепь. Для каждой размерности вероятности побираются индивидуально.

- *Изменчивость разряда*: проверяется, как часто меняется каждый двоичный разряд при представлении цепи в виде переходной последовательности.

В работе [8] утверждается, что последний показатель в оптимальной цепи должен быть стабилен, то есть меняться медленно от итерации к итерации.

В разделе 13 будет показано, что для многих (n, k)-цепей при $k > 2$ первый критерий работает хорошо, второй — плохо.

11.2 Эвристические алгоритмы

Имеется много работ, посвященных решению данной задачи методами дискретной опимизации. Не углубляясь в детали, кратко охарактеризуем некоторые из них.

1. *Муравьиный алгоритм* (S. Hardas, 2005г.)

   Муравьиный алгоритм моделирует поведение поиска путей муравьями и тем самым естественно подходит для данной задачи. Когда колонии реальных муравьев предоставляется доступ к источнику пищи по различным дорогам различной длины, их движение быстро сходится на кратчайшем пути от гнезда до источника пищи. На Рис. 3 показана сходимость муравьев на более коротком пути.

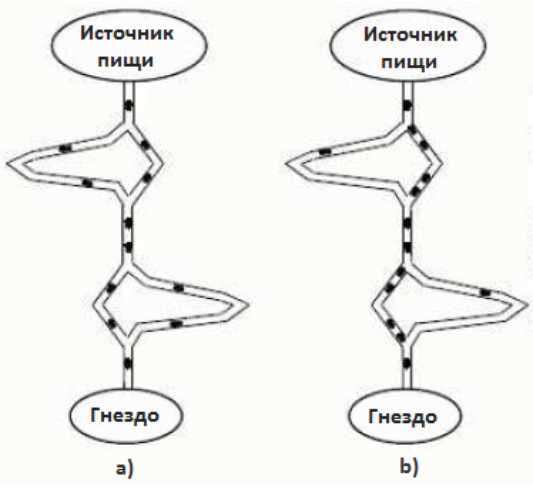

Рис. 3: a)Муравьи начинают исследовать два пути; b)в конечном счете большинство из них выбирает кратчайший путь.

   Муравьи продолжают поиск, пока не найдут источник пищи. Как только они находят его, то кладут химическое вещество (феромон) на обратном пути в гнездо, чтобы остальные знали, где еда. Другие муравьи следуют по пути с самым сильным феромоном. Муравьи, выбирающие более короткий путь, могут совершить больше рейсов за то же время,

что и другие муравьи. Поэтому на более коротких путях будет сосредоточено больше феромона.

Наиболее важным фактором, который влияет на отказ от плохих путей, является испарение феромона. Это происходит естественно, подобно рассеиванию запахов в воздухе. Испарение гарантирует, что в течение времени муравьи забывают пути, которые не были усилены. Таким образом, колония вместе удаляется от более длинных путей.

В случае задачи поиска максимальной цепи некоторое количество «муравьев» строит последовательные решения, перемещаясь от узла гиперкуба к соседнему. Когда у муравья больше нет доступных узлов, он возвращает решение и начинает с узла 0. Когда все муравьи исчерпали свои возможные узлы, считается, что итерация завершилась. Затем муравьи сбросят свои пути, чтобы начать снова с узла 0.

Чтобы строилось корректное решение, нужно, чтобы каждый совершаемый шаг муравья являлся допустимым узлом для цепи, поэтому каждый муравей хранит в памяти посещенные узлы. Он может перемещаться только в другие узлы, смежные с данным и не являющиеся недопустимо близкими к какому-либо узлу в пути.

2. *Метод восхождения на холм* (D. Tuohy, 2007г.)

   Стохастический метод восхождения на холм развивает популяцию змей от нулевой или небольшой начальной длины до некоторой максимальной. Каждая особь в популяции состоит из последовательности узлов цепи. Эти особи инициализируются либо как цепь нулевой длины, состоящая только из нулевого узла, либо как ранее существовавшая предпочтительная цепь.

   В каждом поколении вычисляется оценка пригодности. Используемая функция пригодности основана на длине и «плотности» цепи. Здесь

плотность — это мера того, сколько узлов осталось в гиперкубе после вычитания всех недоступных для добавления в цепь. Она согласована с плотностью, определенной в пункте 11.1.

Длина является главной целью в каждом поколении, а плотность – упорядочивающим фактором ранжирования отбора внутри поколения. После того, как определяется индивидуальная пригодность каждой цепи в популяции, популяция подвергается ранжированию, основанному на пригодности каждой цепи. Для того, чтобы поддерживать постоянную численность особей на протяжении всего цикла, оставшиеся доступные места в популяции следующего поколения заполняются, начиная с наиболее подходящих особей.

После отбора оператор роста увеличивает каждую цепь в популяции на один шаг в каждом поколении, присоединяя к конечному узлу каждой цепи один из ее смежных узлов, который не был дисквалифицирован путем нахождения в цепи, или будучи смежным с каким-либо узлом, включенным в цепь.

3. *Стохастический лучевой поиск* (S. Meyerson, 2014г.)

   Лучевой поиск – оптимизация алгоритма поиска на графе «лучший - первый», где в качестве кандидатов сохраняется только заданное количество путей. Число путей равно ширине луча. Если сгенерировано больше путей, то худшие будут отброшены.

   Поиск производится на структурированном дереве поиска цепей, на кажом уровне содержащем все допустимые подпути, называемые субрешениями. С каждой итерацией исследуется новый уровень дерева поиска. На каждом уровне каждый новый обнаруженный доступный узел добавляется в путь. Каждое добавление узла к субрешению требует добавления к следующей новой популяции субрешений. Каждое

подрешение в новой популяции оценивается с использованием эвристической функции пригодности: менее подходящие решения (ветви) отбрасываются (отсекаются), чтобы уменьшить количество подрешений до ширины луча.

Как и многие эвристические методы, лучевой поиск не может гарантировать, что будет найдено оптимальное решение.

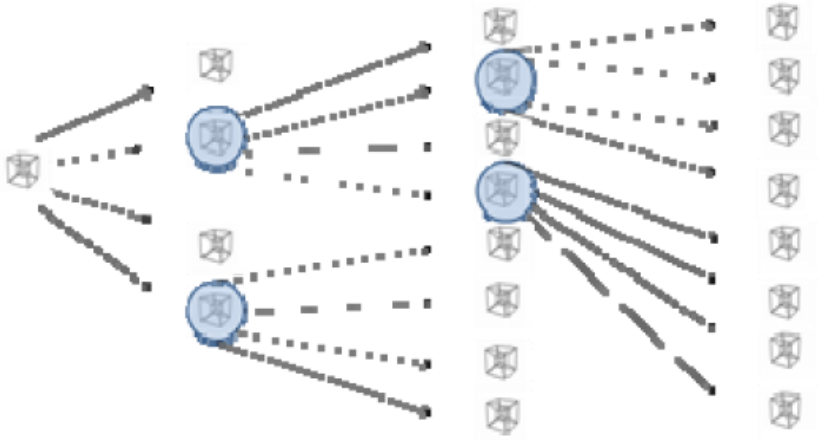

Рис. 4: Лучевой поиск

На Рис. 4 изображена схема лучевого поиска путем последовательного расширения выбранных решений. «Гиперкубы» в синих кружочках представляют решения-кандидаты, которые были выбраны для дальнейшего исследования на следующей итерации. Необведенные решения отбрасываются. Ширина луча в этом случае равна 2.

12 Выбор алгоритма

Для реализации выбран алгоритм, использующий обрезки ветвей по критерию плотности в сочетании с идеями стохастического поиска.

На Рис. 5 приведен псевдокод алгоритма, использующий новые дополнительные структуры данных:

- $numConflictsSpread[x]$: количество вершин в α, находящихся на расстоянии ровно k от x;
- $conflictSpread[x][i]$: массив узлов, находящихся на расстоянии ровно k от каждого x.

Для данного алгоритма написан метод $addVertex()$, использующий наборы вероятностей, возвращающий $true$, когда вершина будет добавлена в путь и $false$, иначе. $P = (P_1, P_2, P_3, P_4)$ — вектор, задающий вероятности. В случае, когда все $P_i = 100$, $i = 1..4$, алгоритм станет алгоритмом исчерпывающего поиска.

Добавленные блоки кода к псевдокоду, приведенному в разделе 7, выделены синим цветом.

```
void Search (int d) {
int x; int y; int p ={последний элемент текущего пути}
    {вычисляется минимальная (min), средняя (average) и
    максимальная (max) плотность кандидатов на добавление}
    for (int i = 0; i ≤ min(d + 1, n − 1); i + +) {
        {x получен изменением i-го бита элемента p}
        if (numConflicts[x] < k) &&
        ((numConflictsSpread[x] == average)||{addVertex()}) {
            α_t = x;
            for each y ∈ conflict[x] do numConflicts[y] + +;
            for each y ∈ conflictSpread[x] do
                numConflictsSpread[y] + +;
            Search(max(i, d));
        }
    }
    for each y ∈ conflict[x] do numConflicts[y] − −;
    for each y ∈ conflictSpread[x] do numConflictsSpread[y] − −;
    {удаление из пути последнего элемента}
}
```

Рис. 5: Псевдокод алгоритма раздела 12

В Таблице 1 в скобках указаны наборы вероятностей добавления вершины в цепь, если ее плотность x попадает в один из четырех промежутков:

1. $min \leq x < (average + min)/2)$,
2. $(average + min)/2 \leq x < average$,
3. $average \leq x < (average + max)/2$,
4. $(average + max)/2 \leq x \leq max$

Каждому i-му промежутку соответствует P_i.

13 Численные результаты

В этом разделе будут приведены результаты вычислений с помощью разных методов и их сравнение. А результаты, полученные методом незаконченного полного перебора, будут использоваться для сравнения.

Программы для расчетов написаны на языке Java. Расчеты проводились для $n = 5..15$ и $k = 2..9$. Вычисления производились на компьютере с процессором intel core i5 3210m. Текст программы находится по ссылке https://yadi.sk/d/pstEP6aB3JRUdY/2017.

13.1 Количество и длины найденных цепей

В Таблице 1 приведены некоторые результаты для сравнения работы трех вариантов алгоритма. Для каждой пары (n, k) в первой строке – наилучший результат, во второй – наихудший.

	Исчерпывающий перебор (прерван после 5 минут)		Стохастическая обрезка P=(30, 40, 60, 100)		Стохастическая обрезка P=(5, 15, 20, 100)	
(n,k)	Длина цепей	Число цепей	Длина цепей	Число цепей	Длина цепей	Число цепей
(7,2)	47	15	50	2	49	2
			48	18	49	1
(11,2)	411	15536	478	3	499	4
			446	1	473	7
(15,5)	151	18	163	70	159	3
			154	1	149	8

Таблица 1: Сравнение результатов после 5 минут работы программ

Как видно, алгоритм стохастической обрезки строит более длинные цепи, чем исчерпывающий перебор, за то же время. Но для разных пар (n, k) наборы вероятностей, дающих лучшие результаты, различны. Поэтому имеет смысл запускать стохастическую обрезку с разными вероятностями.

13.2 Время поиска первой максимальной цепи

В Таблице 2 проведено сравнение, через сколько минут после запуска программы нашлась первая цепь максимальной длины для данной пары (n, k). Аналогично пункту 13.1, в первой строке – наилучший полученный результат, во второй – наихудший.

(n,k)	Длина цепей	Исчерпывающий перебор	Стохастическая обрезка, P=(10 20 30 100)
(7,2)	50	125 мин.	54 мин.
			цепь длины 50 не найдена
(8,3)	35	16 мин.	0.07 мин.
			0.2 мин.
(12,6)	33	13 мин.	0.2 мин.
			цепь длины 33 не найдена

Таблица 2: Сравнение времени нахождения максимальных цепей, в минутах

Можно сделать вывод, что стохастическая обрезка может строить цепи той же длины за намного меньшее время, но иногда вообще не находит. В этом случае цепи, близкие к максимальным, находятся достаточно быстро.

13.3 Изменчивость разрядов максимальных цепей

Для проверки утверждения D. Tuohy о стабильности показателя изменчивости разрядов (раздел 11.1) написана программа, которая для всех максимальных цепей (в размерностях, где исчерпывающий поиск был завершен) вычисляет изменчивость разряда. Как показали результаты, наименьший разряд используется реже всего. Кроме того, как правило, чем больше разряд, тем чаще он используется. Например, для размерности $(9, 4)$, для которой найдено 116 максимальных цепей длины 28, массив изменчивости разрядов имеет вид, приближенный к $[5, 4, 3, 2, 4, 3, 2, 3, 1]$, здесь старший разряд меняется 5 раз. Но есть и исключения: $[3, 4, 5, 4, 2, 3, 3, 2, 1]$, $[2, 3, 5, 4, 3, 2, 3, 3, 2]$.

Для $(7, 3)$-цепей результат наблюдений не такой хороший. Всего име-

ется 4 цепи максимальной длины 21, их массивы изменчивости разрядов: $[5,4,2,3,3,2,1]$, $[4,5,3,2,3,2,1]$, $[4,2,3,5,3,2,1]$, $[2,4,5,3,2,3,1]$. Описанные выше закономерности свойственны лишь первой цепи.

Итак, критерий D. Tuohy признан нерабочим для $k > 2$. Его можно применять лишь в случае, когда один из разрядов поменялся слишком много раз, тогда его больше не меняют.

13.4 Сводная таблица результатов

В Таблице 3 собраны наилучшие результаты, которых удалось достичь с помощью двух запрограммированных алгоритмов. Используемые обозначения: a – найдено путем частичного исчерпывающего поиска (если время счета было критически велико, то поиск обрывался); b – найдено путем стохастической обрезки ветвей; отсутствие буквы – путем полного исчерпывающего поиска (длина максимальна). Цветом выделены наилучшие ранее известные значения; в случае $\sim$ значение наилучшее, но максимальность не гарантирована.

Из таблицы видно, что для многих пар (n,k) стохастическая обрезка дает тот же или лучший результат.

		k							
		2	3	4	5	6	7	8	9
n	5	13 13	7 7	6 6	5 5	5 5	5 5	5	5
	6	26 26	13 13	8 8	7 7	6 6	6 6	6	6
	7	50a 50b ∼50	21 21	11 11	9 9	8 8	7 7	7	7
	8	85a 89b ∼98	35a 35b 35	19 19	11 11	10 10	9 9	8	8
	9	147a 155b ∼190	55a 57b ∼63	28 28	19 19	12 12	11 11	10	9
	10	254a 282b ∼370	89a 89b ∼103	43a 45b 47	25 25	15 15	13 13	12	11
	11	416a 499b ∼695	136a 141b ∼157	65a 66b ∼68	38a 38b 39	25 25	15 15	14	13
	12	776a 882b ∼1274	230a 231b ∼286	94a 96b ∼104	53a 53b ∼56	33 33	25 25	16	15
	13	1466a 1628b ∼2466	377a 372b ∼493	146a 152b ∼181	77a 77b ∼79	46a 46b ∼47	31 31	19	17
	14	2711a 2972b ∼4932	627a 615b ∼811	212a 220b ∼279	106a 106b ∼112	63a 62b ∼66	40a 40b ∼42	31	19
	15	5217a 5594b ∼9866	1034a 1054b ∼1379	339a 358b ∼480	151a 163b ∼206	89a 87b ∼89	53a 53b∼55	36a 36b	31

Таблица 3: Найденные длины (n, k)-цепей

14 Заключение

В работе были написаны две программы: одна (раздел 7) реализует алгоритм исчерпывающего поиска в усеченном пространстве решений, вторая (раздел 12) — алгоритм стохастической обрезки ветвей. По причине недостаточной вычислительной мощности компьютера вычисления для больших размерностей прерывались после $1,5-2$ часов работы программ.

Проведенные вычисления позволили расширить таблицу известных длин максимальных цепей для $k=8$ и $k=9$.

На основании полученных расчетов и сопоставления результатов алгоритмов можно сделать вывод, что алгоритм обрезки ветвей работает хорошо для поставленной задачи. Но для каждой пары (n,k) векторы вероятностей подбираются индивидуально, что может увеличить время исследования.

Все поставленные работе цели были достигнуты. В дальнейшем хотелось бы добавить в алгоритм еще несколько критериев отказа от неперспективных цепей.

Из обозначенных эвристических методов наиболее перспективным для реализации кажется муравьиный алгоритм, так как он естественно подходит для данной задачи.

К задаче «змея в ящике» продолжают применять все новые и новые методы и подходы, ожидаются новые рекорды.

Список литературы

[1] **Евдокимов А. А.** *Цепные коды и Snake-in-the-Box Problem* — Казань, Ученые записки Казанского университета. Серия Физико-математические науки, 156, № 3, 2014, 55–65.

[2] **Allison D., Paulusma D.** *New Bounds for the Snake-in-the-Box Problem* — United Kingdom: Article, School of Engineering and Computing Sciences, Durham University, 2016.

[3] **Casella D. A., Potter W. D.** *Using evolutionary techniques to hunt for snakes and coils* — USA: Evolutionary Computation, The 2005 IEEE Congress on, 2005, 2499-2505.

[4] **Hardas S. P.** *An ant colony approach to the Snake-in-the-Box problem* — USA: Master of Science Thesis, The University of Georgia, 2005.

[5] **Hood S., Recoskie D., Sawada J.** *Snakes, coils, and single-track circuit codes with spread k* — USA: Journal of Combinatorial Optimization, 2015, 42-62.

[6] **Kochut K. J.** *Snake-In-The-Box Codes for Dimension 7, PhD dissertation* — USA: University of Georgia, 2009.

[7] **Meyerson S., Drapela T., Potter W., Whiteside W.** *Finding longest paths in hypercubes, snakes and coils* — USA: IEEE Symposium on Computational Intelligence for Engineering Solutions, 2014.

[8] **Tuohy D. R., Potter W. D., Casella, D. A.** *Searching for Snake-in-the-Box Codes with Evolved Pruning Models* — USA: Proceedings of the 2007 Int. Conf. on Genetic and Evolutionary Methods, 2007, 3–9.